Checkpoint Space Reclamation for Uncoordinated Checkpointing in Message-Passing Systems

Yi-Min Wang, Pi-Yu Chung, In-Jen Lin and W. Kent Fuchs

Abstract— **Uncoordinated checkpointing allows process autonomy and general nondeterministic execution, but suffers from potential domino effects and the associated space overhead. Previous to this research, checkpoint space reclamation had been based on the notion of obsolete checkpoints; as a result, a potentially unbounded number of nonobsolete checkpoints may have to be retained on stable storage. In this paper, we derive a necessary and sufficient condition for identifying all garbage checkpoints. By using the approach of recovery line transformation and decomposition, we develop an optimal checkpoint space reclamation algorithm and show that the space overhead for uncoordinated checkpointing is in fact bounded by $N(N+1)/2$ checkpoints where N is the number of processes.**

Keywords— **fault tolerance, message-passing systems, uncoordinated checkpointing, rollback recovery, garbage collection.**

I. Introduction

Checkpointing and rollback recovery is an effective approach to tolerating both hardware and software faults. During normal execution, the state of each process is periodically saved on stable storage as a *checkpoint*. When a failure occurs, the process can roll back to a previous checkpoint by reloading the checkpointed state. In a message-passing system, *rollback propagation* can occur when the rollback of a message sender results in the rollback of the corresponding receiver. The system is then required to roll back to the latest available consistent set of checkpoints called the *recovery line* to ensure correct recovery with a minimum amount of rollback. In the worst case, cascading rollback propagation may result in the *domino effect* [1] which prevents recovery line progression.

Numerous checkpointing and rollback recovery techniques have been proposed in the literature for message-passing systems. *Uncoordinated checkpointing* [2-4] allows maximum process autonomy and general nondeterministic execution. Each process takes its checkpoints independently and keeps track of the dependencies among checkpoints resulting from message communications. When a failure occurs, the dependency information is used to determine the recovery line to which the system should roll back. The major disadvantages of uncoordinated checkpointing have been the potential domino effect and the space overhead required for maintaining multiple checkpoints of each process.

This paper addresses the second disadvantage by developing an optimal checkpoint space reclamation algorithm to minimize the space overhead. Several techniques have been proposed to address the first issue, i.e., to guarantee recovery line progression. *Coordinated checkpointing* [5,6] eliminates the domino effect by sacrificing a certain degree of process autonomy. Extra coordination messages are required to enforce the consistency between the checkpoints belonging to the same checkpointing session. The run-time overhead can be reduced if certain optimization techniques can be employed [7]. For applications which require process autonomy in taking checkpoints in order to exploit application-dependent information to checkpoint at the "right time", e.g., when the process state is minimal, *lazy checkpoint coordination* [8] can be incorporated into an uncoordinated checkpointing protocol to provide a trade-off between coordination overhead and recovery efficiency.

Another approach to eliminating the domino effect is to exploit the *piecewise deterministic execution model* [9-11], in which each process execution is viewed as a number of deterministic *state intervals* bounded by nondeterministic events. It has been shown [12] that by considering each nondeterministic event log as a *logical checkpoint* [13] taken at the end of the ensuing state interval, the same dependency model and hence the checkpoint space reclamation algorithm developed in this paper can still be applied.

Traditionally, checkpoint space reclamation for uncoordinated checkpointing has been based on the notion of *obsolete checkpoints*: the global recovery line which suffices to recover from the failure of the entire system is computed; then all of the obsolete checkpoints before that recovery line are no longer useful and can be discarded. In contrast, all of the nonobsolete checkpoints have been assumed to be possibly useful for some future recovery and should be retained. With the possibility of domino effects, the number of nonobsolete checkpoints is potentially unbounded.

Motivated by the observation that being obsolete is simply a sufficient condition for being garbage, we derive a necessary and sufficient condition for identifying all garbage checkpoints, which leads to an optimal checkpoint space reclamation algorithm and the least upper bound on the number of nongarbage checkpoints. Our approach is to model consistent global checkpoints as maximum-sized antichains of the partially ordered set generated by the *happened before* relation between the checkpoints. We define a recovery line transformation and decomposition, and demonstrate that any nongarbage checkpoint belonging to a possible future recovery line must also be contained in one of the N "immediate future" recovery lines, where N is the number of processes. It is also shown that these N recovery lines can contain at most $N(N+1)/2$ distinct nongarbage checkpoints.

The outline of the paper is as follows. Section II describes the checkpointing and recovery protocol and a model of consistent global checkpoints; Section III derives a necessary and sufficient condition for identifying all nongarbage checkpoints and presents the optimal checkpoint space reclamation algorithm; the least upper bound on the number of nongarbage checkpoints is derived in Section IV and experimental evaluation is described in Section V. Due to space limitation, some proofs are omitted and can be found in the complete technical report [14].

II. Checkpointing and Rollback Recovery

A. System Model and Recovery Protocol

The system considered in this paper consists of a number of concurrent processes for which all process communication is through message passing. Processes are assumed to run on fail-

This research was supported in part by the Department of the Navy and managed by the Office of the Chief of Naval Research under Contract N00014-91-J-1283, and in part by the National Aeronautics and Space Administration (NASA) under Grant NASA NAG 1-613, in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS).

Yi-Min Wang was with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801; he is now with AT&T Bell Laboratories, Murray Hill, NJ 07974.

Pi-Yu Chung and W. Kent Fuchs are with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

In-Jen Lin is with the Department of Mathematics, University of Illinois, Urbana, IL 61801.

stop processors [15] and, for the purpose of presentation, each process is considered as an individual *recovery unit*. In order to allow general nondeterministic execution, we do not assume the piecewise deterministic model. This implies that whenever the sender of a message m rolls back and *unsends* m, the receiver which has already processed m must also roll back to undo the effect of m because the potential nondeterminism preceding the sending of m may prevent the same message from being resent during reexecution. Let $c_{i,x}$ denote the xth checkpoint ($x \geq 0$) of process p_i ($0 \leq i \leq N-1$), where N is the number of processes in the system. Two checkpoints $c_{i,x}$ and $c_{j,y}$ are then considered *inconsistent* if there is any message sent after $c_{j,y}$ and processed before $c_{i,x}$, or vice versa. In contrast, when the receiver of a message m' rolls back and *unreceives* m', the sender needs not roll back to *unsend* m' if m' can be retrieved from a synchronous[1] message log [16,17] or through a reliable end-to-end transmission protocol [6].

During normal execution, each process takes its *local checkpoints* periodically without coordinating with any other processes. Let (i,x) denote the xth *checkpoint interval* of process p_i between consecutive checkpoints $c_{i,x}$ and $c_{i,x+1}$. Each message is tagged with the current checkpoint interval number and the process number of the sender. Each receiver p_i performs *direct dependency tracking* [2,18] as follows: if a message sent from (j,y) is processed in (i,x), then the direct dependency of $c_{i,x+1}$ on $c_{j,y}$ is recorded.

A garbage collection procedure can be periodically invoked by any process p_i to reclaim the storage space of garbage checkpoints. First, p_i collects the direct dependency information from all the other processes to construct the *checkpoint graph* [2] in which each vertex represents a checkpoint and each edge represents a direct dependency (including the implicit dependency of any $c_{j,y+1}$ on $c_{j,y}$), as shown in Fig. 1(b). Then the *rollback propagation algorithm* listed in Fig. 2 is executed on the checkpoint graph to determine the *global recovery line*[2] (black vertices), before which all the checkpoints are obsolete (marked "X") and can be discarded.

Fig. 1. Checkpointing and rollback recovery. (a) Example checkpoint and communication pattern; (b) checkpoint graph and extended checkpoint graph when p_0 initiates a rollback.

Fig. 2. The rollback propagation algorithm.

When any process initiates a rollback, it starts a similar procedure for recovery. The current volatile states of the surviving processes are treated as additional *virtual checkpoints* [3] for constructing an *extended checkpoint graph* of which the recovery line is called the *local recovery line* (shaded vertices) and indicates the consistent rollback state.

B. A Model of Consistent Global Checkpoints

In a message-passing system, event e_1 *directly happened before* event e_2 [19] if

- e_1 and e_2 are events in the same process and e_1 occurs immediately before e_2; or
- e_1 is the sending of a message m and e_2 is the receiving of m.

The transitive closure of the *direct happened before* relation is the *happened before* relation, denoted by $<$. The set of events with the *happened before* relation forms a *partially ordered set*, or *poset* [19]. For our purpose, we consider only the induced subposet $R = (C, <)$, where C is the set of all checkpoints.

For a system with N processes, a *global checkpoint* is defined as a set of N local checkpoints, one from each process. Based on the earlier description of consistency, a *consistent global checkpoint* is a global checkpoint of which no two constituent checkpoints are ordered by the *happened before* relation. For the purpose of recovery, we are interested in finding the latest available consistent global checkpoint, referred to as the *recovery line*, which minimizes the total rollback distance.

Our approach is based on the maximum-sized antichain model for consistent global checkpoints [18]. Given a poset $P = (S, <)$, an *antichain* is a subset A of S such that $x \not< y$ for any $x, y \in A$. Intuitively, a consistent global checkpoint corresponds to an antichain of the poset $R = (C, <)$. Since the initial checkpoints of all processes must form an antichain of size N and no antichain can contain two checkpoints from the same process, the largest size of any antichain in R is exactly N. The following lemma summarizes the main results described by Wang *et al.* [18].

Lemma 1: *Given the poset $R = (C, <)$ of checkpoints generated by the happened before relation and $M, M_1, M_2 \subseteq C$, let $\mathcal{M}(R)$ denote the set of maximum-sized antichains of R.*

(a) *M is a consistent global checkpoint if and only if $M \in \mathcal{M}(R)$.*

(b) *Let $M[i]$ denote the constituent checkpoint of M which is a checkpoint of p_i and, for any $M_1, M_2 \in \mathcal{M}(R)$, define $M_1 \preceq M_2$ if $M_1[i] \leq M_2[i]$ for all $0 \leq i \leq N-1$. Then, the poset $(\mathcal{M}(R), \preceq)$ forms a lattice.*

(c) *The recovery line is the unique maximal maximum-sized antichain, denoted by $M^*(R)$, on the lattice $(\mathcal{M}(R), \preceq)$.*

In this paper, we will use the notation $\mathcal{M}(G)$ to represent the set of maximum-sized antichains of the poset corresponding to the transitive closure of the checkpoint graph G.[3] The notation $M^*(G)$ is similarly defined.

III. Optimal Checkpoint Space Reclamation

A. Motivation and Problem Formulation

Since a future program execution may contain arbitrary checkpoint dependencies and rollbacks, we first describe an execution model to make the problem tractable. An *operational session* [3] is the interval between the start of normal execution and the instance of rollback initiation, as shown in Fig. 3. A *recovery session* immediately follows the previous operational session and ends at the resumption of normal execution. A program execution can be viewed as consisting of a number of alternating operational sessions and recovery sessions. In terms of the effect on the checkpoint graphs, new vertices are added as new checkpoints are taken during an operational session, and existing vertices can be deleted as some checkpoints are invalidated by the rollback during a recovery session.

Fig. 3. Operational sessions, recovery sessions and nongarbage checkpoints.

Since the purpose of maintaining checkpoints is for possible future recovery, a checkpoint is *garbage* if and only if it can

[1]Extension of our work to an asynchronous logging protocol is considered elsewhere [4].

[2]The global recovery line is to be used when the entire system fails, while a local recovery line is computed when only a subset of processes fail.

[3]It has been pointed out [18] that the poset R' corresponding to the transitive closure of the checkpoint graph based on direct dependency tracking is not exactly the same as R. However, R' and R possess the same set of maximum-sized antichains.

not belong to any future recovery line. Being obsolete, i.e., before the global recovery line, is simply a sufficient condition for being garbage, but not a necessary condition. We first give an example of nonobsolete garbage checkpoints. Figure 4 is a typical example illustrating the domino effect. The global recovery line stays at the set of initial checkpoints and is unable to move forward. The edge from $c_{0,2}$ to $c_{1,2}$ and the one from $c_{1,1}$ to $c_{0,2}$ imply that $c_{0,2}$ is inconsistent with any checkpoint of process p_1. Since a recovery line must contain one checkpoint from each process, $c_{0,2}$ can not belong to any future recovery line[4] and is therefore a garbage checkpoint. Checkpoints $c_{1,1}$ and $c_{0,1}$ are garbage by similar arguments.

Fig. 4. Example of nonobsolete garbage checkpoints.

Figure 4 in fact provides another sufficient condition for identifying garbage checkpoints; our optimal garbage collection aims at deriving the necessary and sufficient condition. The difficulty of the problem lies in the fact that future process execution may contain any number of operational sessions (with arbitrary checkpoint dependencies) and recovery sessions (with arbitrary subsets of processes being faulty). We outline our approach as follows. Instead of trying to find garbage checkpoints, we start with identifying nongarbage checkpoints. Given any possible future recovery line which contains some nongarbage checkpoints, for example, the recovery line shown in Fig. 3. we perform *recovery line transformation* to transform it into another recovery line which also contains those nongarbage checkpoints. We show that all possible future recovery lines containing any nongarbage checkpoint can be transformed into a set of 2^N "immediate future" recovery lines. (Recall that N is the number of processes.) Our next step is *recovery line decomposition*. We identify a set of N recovery lines which forms the "basis" for those 2^N recovery lines and therefore contains all of the nongarbage checkpoints.

B. Recovery Line Transformation

Our approach to transforming an arbitrary future recovery line backwards in time is to first define two elementary transformations: *transformation within an operational session* and *transformation across a recovery session*. Any transformation can then be achieved through a combination of these two elementary transformations.

Transformation within an operational session

During normal process execution, the size of the checkpoint graph increases as new checkpoints are taken. Because checkpoint graphs represent program dependencies and are not arbitrary directed acyclic graphs, the following rules must be satisfied when adding new vertices. For every new vertex $c_{i,x}$ with $x \geq 1$,

Rule 1.a: $c_{i,x}$ must have an incoming edge from $c_{i,x-1}$;

Rule 1.b: $c_{i,x}$ can not have any outgoing edge to any existing vertices because it can not *happen before* a checkpoint that was taken earlier.

We use $\mathcal{G}_s(G)$ to denote the set of all *potential supergraphs* obtainable by adjoining new vertices to a given checkpoint graph G without violating Rule 1.a and Rule 1.b.

Our transformation procedure generally involves changing part of the recovery line of a graph G_1 to obtain the recovery line of another graph G_2. The following lemma will be used throughout this paper to ensure that the unchanged part, which forms an antichain in G_1, remains an antichain in G_2 after the transformation.

[4]It is not hard to see that $c_{0,2}$ being a garbage checkpoint will not be affected by the occurrence of any recovery session because every rollback either preserves the "triangular" condition in Fig. 4 for $c_{0,2}$ or simply invalidates $c_{0,2}$.

Lemma 2: *Given a checkpoint graph $G = (V, E)$ and its potential supergraph $G' = (V', E') \in \mathcal{G}_s(G)$, for any $A \subseteq V$, A is an antichain in G if and only if A is an antichain in G'.*

One special potential supergraph of G, denoted by $\hat{G}$, will play a major role throughout this paper. The graph $\hat{G}$ is constructed by adjoining a new vertex n_i at the end of G for each p_i, with a single incoming edge from the last vertex l_i, as shown in Fig. 5. Let L denote the set of all *last-nodes* l_i and B denote the set of all *new-nodes* n_i. We will refer to the 2^N graphs $\hat{G} - W$, $W \subseteq B$, as the *immediate supergraphs* of G. The proof of the following property defines the **recovery line transformation within an operational session**:

> given the recovery line of a potential supergraph G' of G, by replacing its constituent checkpoints *which are not contained in* G with their corresponding *new-nodes* of G, we obtain the recovery line of an immediate supergraph of G.

Fig. 5. Construction of the potential supergraph $\hat{G}$.

Property 1: *For any checkpoint v in a checkpoint graph G, if v belongs to the recovery line of a potential supergraph G', then v must also belong to the recovery line of an immediate supergraph of G. That is, given $G = (V, E)$, $v \in V$ and $G' \in \mathcal{G}_s(G)$, if $v \in M^*(G')$, then $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$.*

Proof. We partition $M^*(G')$ into $M_1 \cup M_2$ where $M_1 = M^*(G') \cap V$ and $M_2 = M^*(G') \setminus V$, as shown in Fig. 6. A corresponding partition of the *new-nodes* of G is given as $B = B_1 \cup B_2$ such that $B_1 = \{n_i : M^*(G')[i] \in M_1\}$ and $B_2 = \{n_j : M^*(G')[j] \in M_2\}$. Our goal is to show that $M^*(\hat{G} - B_1) = M_1 \cup B_2$. Then, for any $v \in V$ and $v \in M^*(G')$, we must have $v \in M_1 \subseteq M^*(\hat{G} - W)$ where $W = B_1 \subseteq B$.

Fig. 6. Recovery line transformation within an operational session.

First we show that $M_1 \cup B_2 \in \mathcal{M}(\hat{G} - B_1)$. Define the subset L_2 of *last-nodes* corresponding to M_2 as $L_2 = \{l_j : M^*(G')[j] \in M_2\}$. Because $M_1 \cup L_2$ forms an antichain in G', we must have $M^*(G')[i] \not\leq l_j$ for any $M^*(G')[i] \in M_1$ and $l_j \in L_2$. Now consider $\hat{G} -$ [illegible] we have $M^*(G')[i] \not\leq n_j$ for any $n_j \in B_2$ because each n_j has only a single incoming edge from l_j. Clearly, any *new-node* $n_j \not\leq M^*(G')[i]$. Lemma 2 further guarantees that $M_1 (\subseteq V)$ remains an antichain in G and also in $\hat{G} - B_1$. Hence, we have $M_1 \cup B_2 \in \mathcal{M}(\hat{G} - B_1)$.

We next prove that $M_1 \cup B_2 = M^*(\hat{G} - B_1)$ by contradiction. Suppose $M_1 \cup B_2 \neq M^*(\hat{G} - B_1)$. There must exist $M_1' = M^*(\hat{G} - B_1) \setminus B_2$ such that $M_1' \subseteq V$, $M_1 \preceq M_1'$ and $M_1 \neq M_1'$, as shown in Fig. 6. Now consider G'. Recall that M_1 and M_2 form an antichain in G' and thus for any $u \in M_1'$ and $M^*(G')[j] \in M_2$, we must have $u \not\leq M^*(G')[j]$. We also have $M^*(G')[j] \not\leq u$ by Rule 1.b. Therefore, $M_1' \cup M_2$ forms an antichain in G', contradicting the fact that $M_1 \cup M_2$ is the maximal maximum-sized antichain of G'. □

The transformation within an operational session can be viewed as "projecting" a potential supergraph along the direction opposite to the time axis. It shows that although the number of potential supergraphs of G is infinite, the recovery lines of these graphs can intersect G in only a finite number of ways, and each of the possible intersections must be part of the recovery line of an immediate supergraph of G.

Transformation across a recovery session

Existing vertices on a checkpoint graph, for example, $c_{2,3}$ in Fig. 1(b), can be deleted due to rollback recovery. Let G_E denote the extended checkpoint graph as defined in Section II, $G = (V, E)$ denote the subgraph of G_E without the virtual checkpoints, and $G^- = (V^-, E^-)$ denote the checkpoint graph immediately after recovery. Figure 7 illustrates these checkpoint graphs. Let F denote the part of G deleted by the rollback; then we have $G^- = G - F$. By definition, $M^*(G_E)$ is the local recovery line. Let $M^*(G_E) = M_r \cup M_v$ as shown in Fig. 7(a) where $M_r = M^*(G_E) \cap V$ consists of real checkpoints and $M_v = M^*(G_E) \setminus V$ consists of virtual checkpoints. According to the rollback propagation algorithm, the following two rules must be satisfied when existing vertices are deleted during recovery.

Rule 2.a: There cannot exist any $u \in M_r$ and $w \in V^-$ such that $u < w$, i.e., none of the checkpoints in M_r can have any outgoing edge in G^-;

Rule 2.b: For any u in F, all of the checkpoints reachable by u must also be in F. Consequently, none of the checkpoints in F can have any outgoing edge to any checkpoints in G^-.

Fig. 7. Checkpoint graphs before (G), during (G_E) and after (G^-) recovery.

Property 2 can be proved [14] by defining the **recovery line transformation across a recovery session** as follows:

> given the recovery line M of an immediate supergraph of G^-, for any i such that $M[i]$ *is a new-node and* $M^*(G_E)[i]$ *is not a virtual checkpoint*, we replace $M[i]$ with $M^*(G_E)[i]$ to obtain the recovery line of an immediate supergraph of G.

Property 2: *For any checkpoint v in G^-, if v belongs to the recovery line of an immediate supergraph of G^-, then v must also belong to the recovery line of an immediate supergraph of G. That is, given $G^- = (V^-, E^-)$ and $v \in V^-$, if $v \in M^*(\hat{G}^- - W^-)$ for some $W^- \subseteq B^-$, then $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$, where $\hat{G}^-$, W^- and B^- are defined for G^- in parallel with the definitions of $\hat{G}$, W and B for G, respectively.*

Complete transformation

We now apply Properties 1 and 2 to transforming an arbitrary future recovery line containing any nongarbage checkpoints. By repeatedly applying Property 1 within every operational session and Property 2 across every recovery session, we demonstrate that every such future recovery line of G can be transformed into the recovery line of an immediate supergraph of G which preserves all of those nongarbage checkpoints.

Property 3: [Transformation property] *If a checkpoint in G belongs to a future recovery line, then it must also belong to the recovery line of an immediate supergraph of G. That is, given $G = (V, E)$ and $v \in V$, if $v \in M^*(G')$ for a future checkpoint graph G', then $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$.*

Proof. Without loss of generality, we may assume G is in the qth operational session and G' belongs to the rth session where $r \geq q$. Let G_i denote the checkpoint graph at the *end* of the ith operational session, G_i^- denote the checkpoint graph at the *beginning* of the same session, and W_i denote a subset of *new-nodes* of G_i. Clearly, v must belong to every such intermediate graph. By applying Property 1 to the graph pairs (G', G_r^-), $(G_j - W_j, G_j^-)$ where $q+1 \leq j \leq r-1$ and $(G_q - W_q, G)$, and applying Property 2 to the graph pairs (G_j^-, G_{j-1}) where $q+1 \leq j \leq r$, we can show that v must always remain on the recovery line of an immediate supergraph of one of the intermediate graphs throughout the transformation procedure. Eventually, we have $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$. □

Figure 8 gives an example demonstrating the recovery line transformation. Figure 8(a) is the current checkpoint graph G considered for garbage collection. Suppose that Fig. 8(b) is the extended checkpoint graph when p_3 initiates a rollback, then Fig. 8(c) is the checkpoint graph immediately after the recovery. Figure 8(d) shows another possible extended checkpoint graph when p_0 initiates a second rollback. Since checkpoints A and B are needed for recovery in this case, they should be considered nongarbage checkpoints of G. We first apply Property 1 to the graph pairs (G_d, G_c) and transform the recovery line of G_d into the recovery line of G_g (an immediate supergraph of G_c) by replacing X, Y and Z with their corresponding *new-nodes* of G_c, namely, P, Q and R, respectively. Then we apply Property 2 to the pair (G_c, G_b). Since p_3 and p_4 contribute real checkpoints C and D, respectively, to the local recovery line in Fig. 8(b), the recovery line of G_g is transformed into the recovery line of G_f (an immediate supergraph of G_b) by replacing Q and R with C and D. Finally, by applying Property 1 to the pair (G_f, G), we obtain the recovery line of G_e (an immediate supergraph of G) which still contains the nongarbage checkpoints A and B.

Fig. 8. Example recovery line transformation.

C. Recovery Line Decomposition

Property 3 states that the recovery lines of the 2^N immediate supergraphs of G contain all nongarbage checkpoints. We next show that there exists a set of N recovery lines which forms a "basis" for the 2^N recovery lines: each of the 2^N recovery lines is the set of minimal elements in the union of a subset of the N basis recovery lines. Therefore, it suffices to find these N recovery lines to identify all nongarbage checkpoints.

Let $X \wedge Y$ denote the meet (greatest lower bound) of X and Y in a lattice and $\min(S)$ denote the set of minimal elements in S. Based on the following property from Anderson's book [20]: for any poset Q and $M_1, M_2 \in \mathcal{M}(Q)$, $M_1 \wedge M_2 = \min(M_1 \cup M_2)$, we can show by induction [14] that the greatest lower bound of any k maximum-sized antichains can be obtained as the set of minimal elements in their union.

Lemma 3: *Given a poset P, $M \in \mathcal{M}(P)$ and $M \preceq M_i \in \mathcal{M}(P)$ for $0 \leq i \leq k-1$ for any finite k, define $\bigwedge_{0 \leq i \leq k-1} M_i = (...((M_0 \wedge M_1) \wedge M_2) \;...\;) \wedge M_{k-1}$. Then*

(a) $M \preceq \bigwedge_{0 \leq i \leq k-1} M_i \in \mathcal{M}(P)$;

(b) $\bigwedge_{0 \leq i \leq k-1} M_i = \min(\bigcup_{0 \leq i \leq k-1} M_i)$.

The following lemma which states the relationship between the maximum-sized antichains of G and those of its potential supergraphs is also required for proving the decomposition property.

Lemma 4: *Given a checkpoint graph $G = (V, E)$ and its potential supergraph $G' = (V', E')$, for any $M \subseteq V$,*

(a) *$M \in \mathcal{M}(G)$ if and only if $M \in \mathcal{M}(G')$;*

(b) *$M^*(G) \preceq M^*(G')$;*

(c) *if $M = M^*(G')$ then $M = M^*(G)$.*

Property 4: [Decomposition property] *For every $W \subseteq B$ and $W \neq \emptyset$, $M^*(\hat{G} - W) = \min(\bigcup_{n_i \in W} M^*(\hat{G} - n_i))$.*

Proof. Without loss of generality, let $W = \{n_i : 0 \le i \le k-1\}$ where $1 \le k \le N$. Since $\hat{G} - n_j \in \mathcal{G}_s(\hat{G} - W)$ for all $0 \le j \le k-1$, $M^*(\hat{G} - W) \preceq M^*(\hat{G} - n_j)$ by Lemma 4(b).

Now consider the graph $\hat{G}$. From Lemma 4(a), we have $M^*(\hat{G} - W) \in \mathcal{M}(\hat{G})$ and $M^*(\hat{G} - n_j) \in \mathcal{M}(\hat{G})$ for all $0 \le j \le k-1$. Let $M_k^* = \min(\bigcup_{0 \le j \le k-1} M^*(\hat{G} - n_j))$. From Lemma 3, we have

$$M^*(\hat{G} - W) \preceq \bigwedge_{0 \le j \le k-1} M^*(\hat{G} - n_j) = M_k^* \in \mathcal{M}(\hat{G}).$$

Since $M^*(\hat{G} - n_j)[j] < n_j$ and thus $n_j \notin M_k^*$ for all $0 \le j \le k-1$, every $x \in M_k^*$ must be contained in $\hat{G} - W$. From Lemma 4(a), we have $M_k^* \in \mathcal{M}(\hat{G} - W)$ and hence $M_k^* \preceq M^*(\hat{G} - W)$. Therefore, we have proved that $M^*(\hat{G} - W) = M_k^* = \min(\bigcup_{n_i \in W} M^*(\hat{G} - n_i))$. □

As an example, we demonstrate the decomposition of $M^*(G_e)$ in Fig. 8(e) where $G_e = \hat{G} - \{n_0, n_1, n_3, n_4\}$. From Property 4 and referring to Fig. 9, we have

$$\begin{aligned} M^*(G_e) &= \min\Big(\bigcup_{n_i \in \{n_0, n_1, n_3, n_4\}} M^*(\hat{G} - n_i)\Big) \\ &= \min(\{A, B, n_2, n_3, n_4, n_0, I, n_1, J, C, D\}) \\ &= \{A, B, n_2, C, D\} \end{aligned}$$

which is exactly the recovery line shown in Fig. 8(e).

Fig. 9. Example of the PCSR algorithm. Shaded checkpoints in (a)-(e) belong to the recovery lines and the nonshaded checkpoints in (f) are garbage.

D. *Predictive Checkpoint Space Reclamation Algorithm*

We are now prepared to derive a necessary and sufficient condition for identifying all nongarbage checkpoints.

Theorem 1: *A checkpoint v in a checkpoint graph G is nongarbage if and only if $v \in M^*(\hat{G} - n_i)$ for some $0 \le i \le N-1$.*

Proof. If $v \in M^*(\hat{G} - n_i)$ for some $0 \le i \le N-1$, then v is nongarbage because $\hat{G} - n_i$ is a possible future checkpoint graph of G. Conversely, if v is nongarbage, we have by definition $v \in M^*(G')$ for some future checkpoint graph G'. From Property 3, $v \in M^*(\hat{G} - W)$ for some $W \subseteq B$; from Property 4,

$$\begin{aligned} v &\in \min\Big(\bigcup_{n_i \in W} M^*(\hat{G} - n_i)\Big) \\ &\subseteq \bigcup_{n_i \in W} M^*(\hat{G} - n_i) \subseteq \bigcup_{0 \le i \le N-1} M^*(\hat{G} - n_i). \end{aligned}$$

Therefore, $v \in M^*(\hat{G} - n_i)$ for some $0 \le i \le N-1$. □

Based on Theorem 1 we now present the *Predictive Checkpoint Space Reclamation (PCSR)* algorithm in Fig. 10 for finding the N recovery lines. Since the rollback propagation algorithm in Fig. 2 is of time complexity $O(|E|)$ where $|E|$ is the total number of edges in the checkpoint graph (as every edge visited can be deleted), the PCSR algorithm is of time complexity $O(N|E|)$.

Fig. 10. The Predictive Checkpoint Space Reclamation algorithm.

An example illustrating the execution of the PCSR algorithm on the checkpoint graph G in Fig. 5 is shown in Fig. 9. All of the checkpoints in G are nonobsolete and must be retained according to the traditional algorithm. Our PCSR algorithm, however, determines that all of the nonshaded checkpoints in Fig. 9(f) can be discarded.

IV. Least Upper Bound on Number of Nongarbage Checkpoints

Theorem 1 not only identifies the minimum set of nongarbage checkpoints but also places an upper bound N^2 on the number of nongarbage checkpoints because each $M^*(\hat{G} - n_i)$, $0 \le i \le N-1$, consists of N checkpoints. The following property identifies the inherent relations among $M^*(\hat{G} - n_i)$'s, and is the key to further improving the N^2 upper bound to the least upper bound $N(N+1)/2$.

Property 5: *For any $0 \le i, j \le N-1$ and $i \ne j$, if $M^*(\hat{G} - n_i)[j] \ne n_j$ and $M^*(\hat{G} - n_j)[i] \ne n_i$, then $M^*(\hat{G} - n_i) = M^*(\hat{G} - n_j)$.*

We are now prepared to prove the second major result of this paper.

Theorem 2: *Let $N_g(G)$ denote the set of nongarbage checkpoints of G and N be the number of processes. Then, $|N_g(G)| \le N(N+1)/2$.*

Proof. By Theorem 1, we have to consider only the N^2 vertices $M^*(\hat{G} - n_i)[j]$, $0 \le i, j \le N-1$. First, $M^*(\hat{G} - n_i)[i]$ for all $0 \le i \le N-1$ must be in G and must contribute N vertices to $N_g(G)$. For the remaining $N^2 - N$ vertices with $i \ne j$, we consider the pair $M^*(\hat{G} - n_i)[j]$ and $M^*(\hat{G} - n_j)[i]$ one at a time and there are $(N^2 - N)/2$ such pairs. We distinguish three cases:

Case 1: $M^*(\hat{G} - n_i)[j] = n_j$ and $M^*(\hat{G} - n_j)[i] = n_i$. Both *new-nodes* do not belong to $N_g(G)$.
Case 2: $M^*(\hat{G} - n_i)[j] = n_j$ and $M^*(\hat{G} - n_j)[i] \ne n_i$, or $M^*(\hat{G} - n_i)[j] \ne n_j$ and $M^*(\hat{G} - n_j)[i] = n_i$. This pair will possibly add one new checkpoint to $N_g(G)$.
Case 3: $M^*(\hat{G} - n_i)[j] \ne n_j$ and $M^*(\hat{G} - n_j)[i] \ne n_i$. It follows from Property 5 that $M^*(\hat{G} - n_i) = M^*(\hat{G} - n_j)$, and thus $M^*(\hat{G} - n_i)[j] = M^*(\hat{G} - n_j)[j]$ and $M^*(\hat{G} - n_j)[i] = M^*(\hat{G} - n_i)[i]$. Since $M^*(\hat{G} - n_j)[j]$ and $M^*(\hat{G} - n_i)[i]$ are already in $N_g(G)$, this case does not contribute any new nongarbage checkpoint.

Therefore, each of the $(N^2 - N)/2$ pairs can contribute at most one new checkpoint to $N_g(G)$ and hence $|N_g(G)| \le N + (N^2 - N)/2 \times 1 = N(N+1)/2$. □

We next show that $N(N+1)/2$ is in fact the least upper bound because for any N we can construct a checkpoint graph G_N^* as shown in Fig. 11 to achieve this upper bound. Figure 11 shows the nongarbage checkpoints contributed by each of the N recovery lines in the PCSR algorithm. All of the $N(N+1)/2$ checkpoints are identified as nongarbage checkpoints.

Fig. 11. G_N^*: The checkpoint graph with $N(N+1)/2$ nongarbage checkpoints.

As a final note, the greatest lower bound of N is achieved when none of the $(N^2 - N)/2$ pairs contributes any nongarbage checkpoint. Coordinated checkpointing protocols [6] guarantee that, immediately after a checkpointing session, the *last-node* of every process must be a maximal element; as a result, Case 1 holds for all pairs, thereby achieving the greatest lower bound.

V. Trace-Driven Simulation Results

Four parallel programs are used to illustrate the checkpoint space reclamation capabilities and benefits of the PCSR algorithm. Two of them are CAD programs written for Intel iPSC/2 hypercube: Cell Placement and Channel Router; the other two are Knight Tour and N-Queen written in the *Chare Kernel* language, which has been developed as a message-driven machine-independent parallel language [21]. We use the Encore Multimax 510 multiprocessor version of the Chare Kernel. Communication traces are collected for these four programs, and trace-driven simulation is performed to obtain the results. The checkpoint interval for each program is arbitrarily chosen to be approximately ten percent of the total execution time, as shown in Table 1.

Table 1. Execution and checkpoint parameters of the programs.

Figure 12 compares our PCSR algorithm with the traditional algorithm for typical executions of the four programs. Each curve shows the number of checkpoints which would be retained if the algorithm is invoked after a certain number of checkpoints have been taken. The domino effect is illustrated by the linear increase in the number of nonobsolete checkpoints as the total number of checkpoints increases. The largest difference between the number of nonobsolete checkpoints and the number of nongarbage checkpoints for each program is 39 versus 7 for Cell Placement, 48 versus 12 for Channel Router, 24 versus 10 for Knight Tour and 41 versus 5 for N-Queen.

Fig. 12. Nonobsolete versus nongarbage checkpoints for the four parallel programs.

VI. Summary

We have derived a necessary and sufficient condition for identifying all garbage checkpoints in an uncoordinated checkpointing protocol. We proved that there exists a set of N recovery lines, where N is the number of processes, such that any checkpoint useful for a possible future recovery must be contained in one of the N recovery lines. An optimal checkpoint space reclamation algorithm of time complexity $O(N|E|)$, where $|E|$ is the number of edges in the checkpoint graph, was presented to identify all nongarbage checkpoints; the storage space for the remaining checkpoints can then be reclaimed. In addition, we demonstrated that the least upper bound on the number of nongarbage checkpoints is $N(N+1)/2$. Communication trace-driven simulation for four parallel programs demonstrated that the algorithm can be effective in significantly reducing the number of retained checkpoints.

Acknowledgement

The authors wish to express their sincere thanks to Douglas West and Weiping Shi for their discussions; to Michael Loui, Andy Lowry, Kang Shin, Robert Strom, and the anonymous referees for their valuable comments; to Balkrishna Ramkumar and Junsheng Long for their help with the experimental results; to L. V. Kalé for access to the Chare Kernel and to Prith Banerjee for use of his hypercube programs.

References

[1] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Software Eng.*, Vol. SE-1, No. 2, pp. 220–232, June 1975.

[2] K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic recovery schemes for distributed processes," in *Proc. IEEE 2nd Symp. on Reliability in Distributed Software and Database Systems*, pp. 124–130, 1981.

[3] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback for recovery - An optimistic approach," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 3–12, 1988.

[4] Y. M. Wang and W. K. Fuchs, "Optimistic message logging for independent checkpointing in message-passing systems," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 147–154, Oct. 1992.

[5] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Trans. Comput. Syst.*, Vol. 3, No. 1, pp. 63–75, Feb. 1985.

[6] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. Software Eng.*, Vol. SE-13, No. 1, pp. 23–31, Jan. 1987.

[7] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The performance of consistent checkpointing," in *Proc. IEEE Symp. Reliable Distributed Syst.*, pp. 39–47, Oct. 1992.

[8] Y. M. Wang and W. K. Fuchs, "Lazy checkpoint coordination for bounding rollback propagation." to appear in *Proc. 12th Symp. on Reliable Distributed Syst.*, Oct. 1993.

[9] R. E. Strom and S. Yemini, "Optimistic recovery in distributed systems," *ACM Trans. Comput. Syst.*, Vol. 3, No. 3, pp. 204–226, Aug. 1985.

[10] D. B. Johnson and W. Zwaenepoel, "Recovery in distributed systems using optimistic message logging and checkpointing," *J. Algorithms*, Vol. 11, pp. 462–491, 1990.

[11] A. P. Sistla and J. L. Welch, "Efficient distributed recovery using message logging," in *Proc. 8th ACM Symposium on Principles of Distributed Computing*, pp. 223–238, 1989.

[12] Y. M. Wang, "Space reclamation for uncoordinated checkpointing in message-passing systems." Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Aug. 1993.

[13] Y. M. Wang, Y. Huang, and W. K. Fuchs, "Progressive retry for software error recovery in distributed systems," in *Proc. IEEE Fault-Tolerant Computing Symp.*, pp. 138–144, June 1993.

[14] Y. M. Wang, P. Y. Chung, I. J. Lin, and W. K. Fuchs, "Checkpoint space reclamation for uncoordinated checkpointing in message-passing systems." Tech. Rep. CRHC-92-06, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1992.

[15] R. D. Schlichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems," *ACM Trans. Comput. Syst.*, Vol. 1, No. 3, pp. 222–238, Aug. 1983.

[16] A. Borg, W. Blau, W. Graetsch, F. Herrmann, and W. Oberle, "Fault tolerance under UNIX," *ACM Trans. Comput. Syst.*, Vol. 7, No. 1, pp. 1–24, Feb. 1989.

[17] M. L. Powell and D. L. Presotto, "Publishing: A reliable broadcast communication mechanism," in *Proc. 9th ACM Symp. Oper. Syst. Principles*, pp. 100–109, 1983.

[18] Y. M. Wang, A. Lowry, and W. K. Fuchs, "Consistent global checkpoints based on direct dependency tracking." Research Report RC 18465, IBM T.J. Watson Research Center, Yorktown Heights, New York, Oct. 1992. Submitted to *Inform. Process. Lett.*

[19] L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Commun. ACM*, Vol. 21, No. 7, pp. 558–565, July 1978.

[20] I. Anderson, *Combinatorics of Finite Sets*. Oxford: Clarendon Press, 1987.

[21] W. Shu and L. V. Kalé, "Chare kernel - A runtime support system for parallel computations," *J. Parallel Distributed Comput.*, Vol. 11, pp. 198–211, 1991.

AFFILIATION OF AUTHORS:

Yi-Min Wang was with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801; he is now with AT&T Bell Laboratories, Murray Hill, NJ 07974.

Pi-Yu Chung and W. Kent Fuchs are with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

In-Jen Lin is with the Department of Mathematics, University of Illinois, Urbana, IL 61801.

ACKNOWLEDGMENT OF FINANCIAL SUPPORT:

This research was supported in part by the Department of the Navy and managed by the Office of the Chief of Naval Research under Contract N00014-91-J-1283, and in part by the National Aeronautics and Space Administration (NASA) under Grant NASA NAG 1-613, in cooperation with the Illinois Computer Laboratcry for Aerospace Systems and Software (ICLASS).

NUMBERED FOOTNOTES:

(1) Extension of our work to an asynchronous logging protocol is considered elsewhere [4].

(2) The global recovery line is to be used when the entire system fails, while a local recovery line is computed when only a subset of processes fail.

(3) It has been pointed out [18] that the poset R' corresponding to the transitive closure of the checkpoint graph based on direct dependency tracking is not exactly the same as R. However, R' and R possess the same set of maximum-sized antichains.

(4) It is not hard to see that $c_{0,2}$ being a garbage checkpoint will not be affected by the occurrence of any recovery session because every rollback either preserves the "triangular" condition in Fig. 4 for $c_{0,2}$ or simply invalidates $c_{0,2}$.

FIGURE CAPTIONS:

Fig. 1. Checkpointing and rollback recovery. (a) Example checkpoint and communication pattern; (b) checkpoint graph and extended checkpoint graph when p_0 initiates a rollback.

Fig. 2. The rollback propagation algorithm.

Fig. 3. Operational sessions, recovery sessions and nongarbage checkpoints.

Fig. 4. Example of nonobsolete garbage checkpoints.

Fig. 5. Construction of the potential supergraph $\hat{G}$.

Fig. 6. Recovery line transformation within an operational session.

Fig. 7. Checkpoint graphs before (G), during (G_E) and after (G^-) recovery.

Fig. 8. Example recovery line transformation.

Fig. 9. Example of the PCSR algorithm. Shaded checkpoints in (a)-(e) belong to the recovery lines and the nonshaded checkpoints in (f) are garbage.

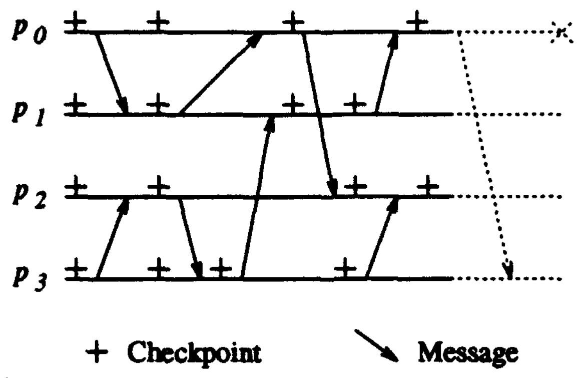
Fig. 10. The Predictive Checkpoint Space Reclamation algorithm.

Fig. 11. G_N^*: The checkpoint graph with $N(N+1)/2$ nongarbage checkpoints.

Fig. 12. Nonobsolete versus nongarbage checkpoints for the four parallel programs.

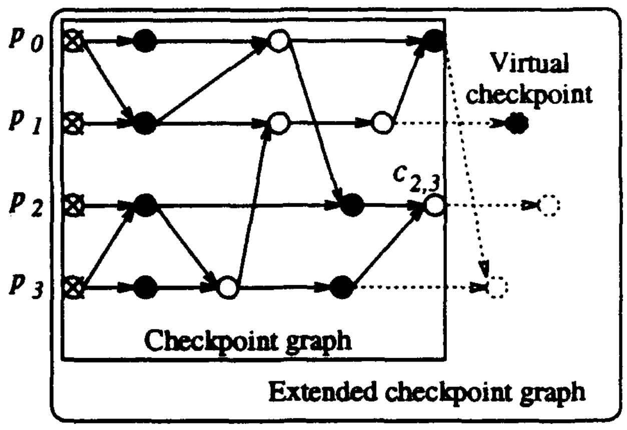
TABLE CAPTION:

Table 1. Execution and checkpoint parameters of the programs.



p_0
p_1
p_2
p_3
+ Checkpoint
Message

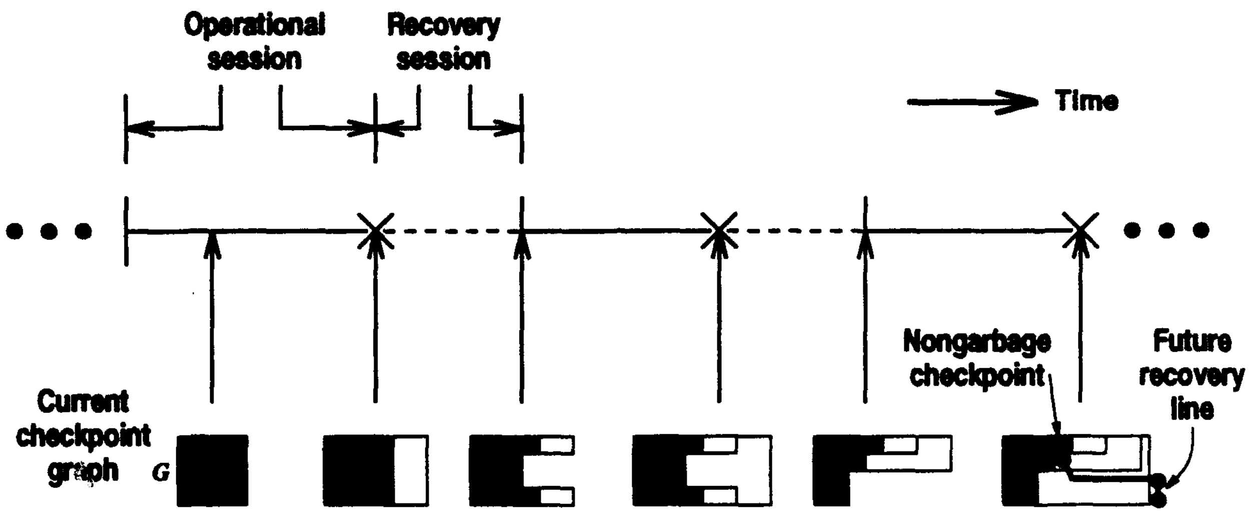
(a)

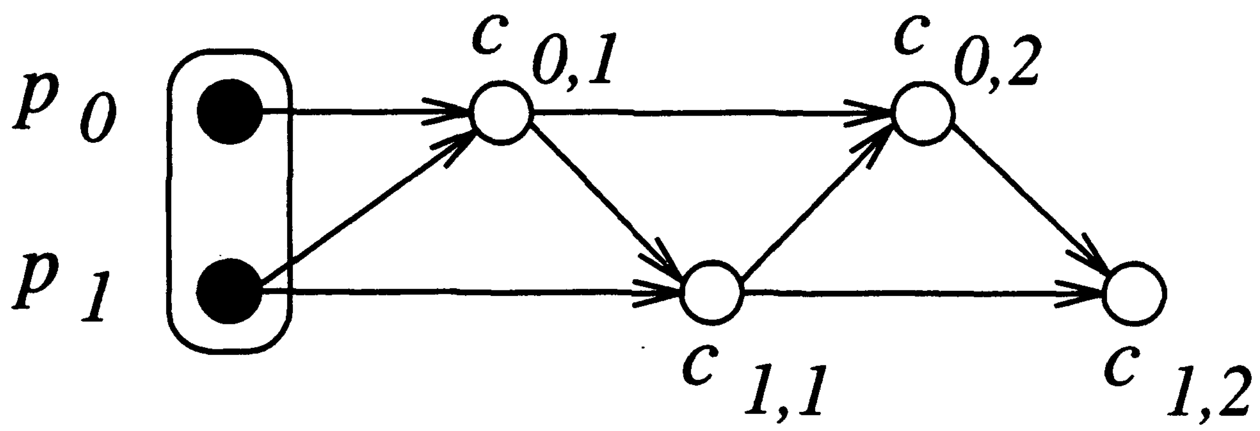


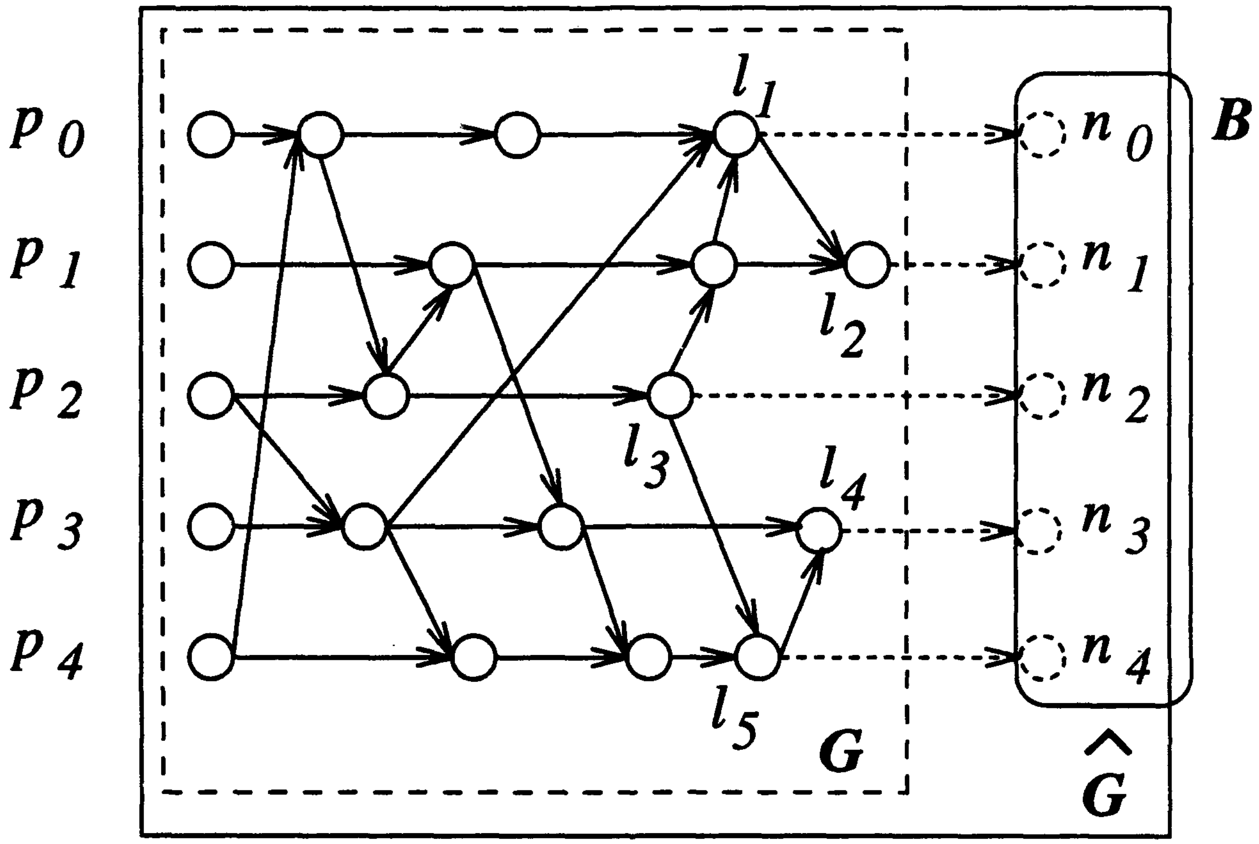
p_0
p_1
p_2
p_3
Virtual checkpoint
$c_{2,3}$
Checkpoint graph
Extended checkpoint graph

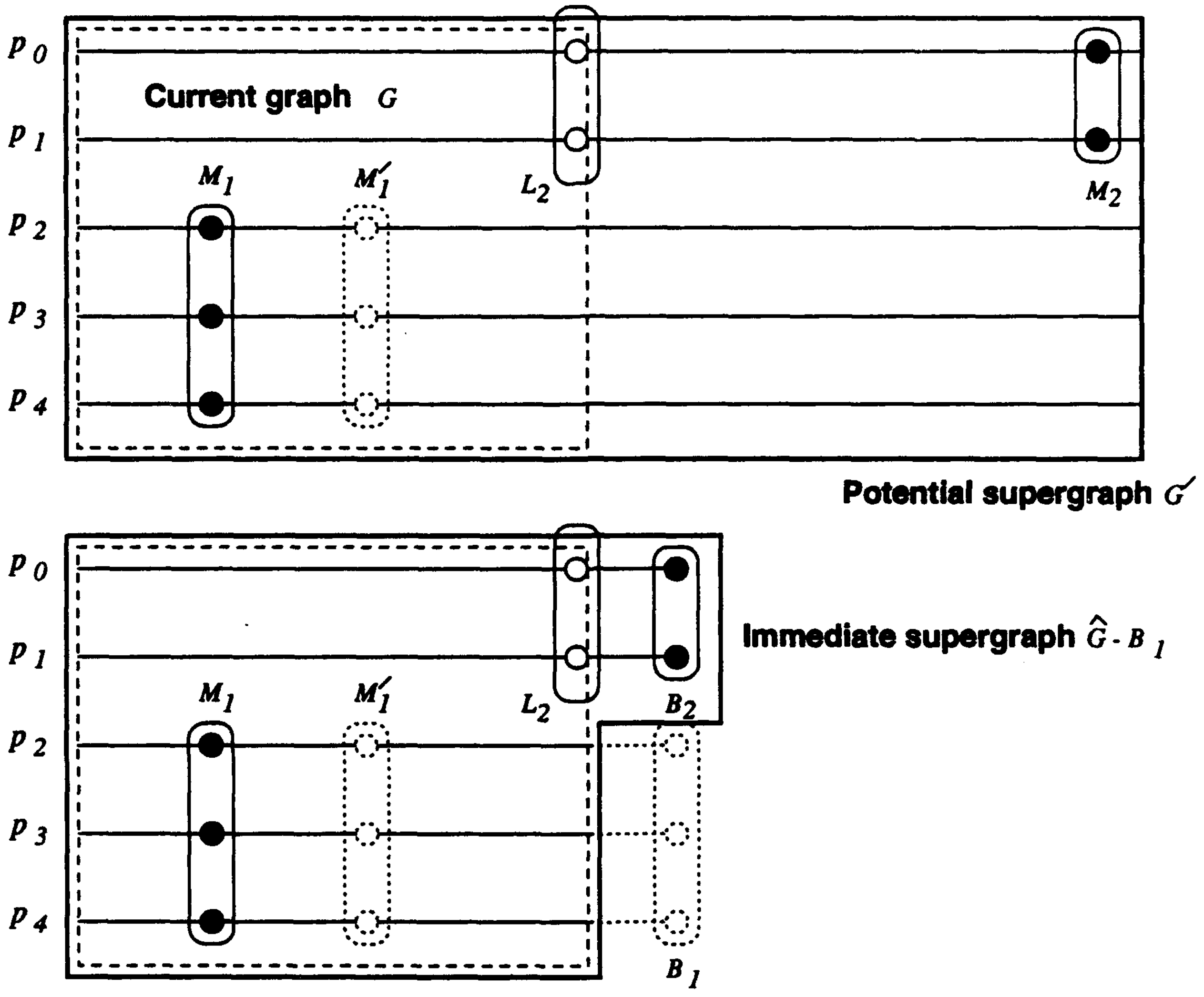
(b)

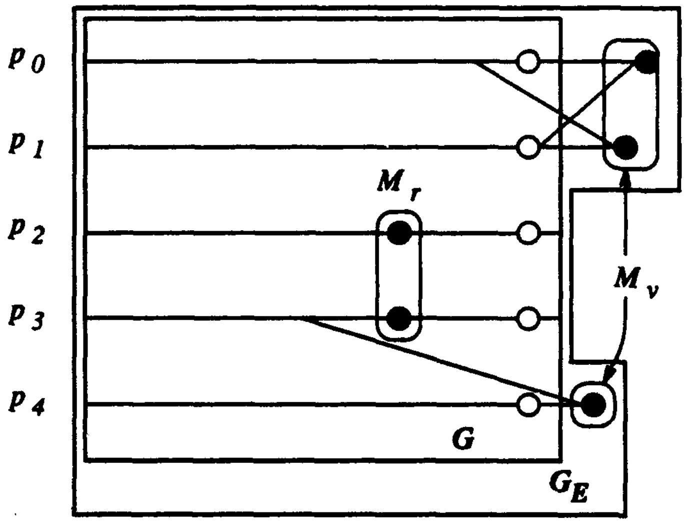
```
/* CP stands for checkpoint */
/* Initially, all of the CPs are unmarked */
include the latest CP of each process in the root set;
mark all CPs strictly reachable from any CP in the root set;
while (at least one CP in the root set is marked) {
    replace each marked CP in the root set by the latest unmarked CP of the same
    process;
    mark all CPs strictly reachable from any CP in the root set;
}
the root set is the recovery line.
```


Operational session
Recovery session
Time
Current checkpoint graph
G
Nongarbage checkpoint
Future recovery line

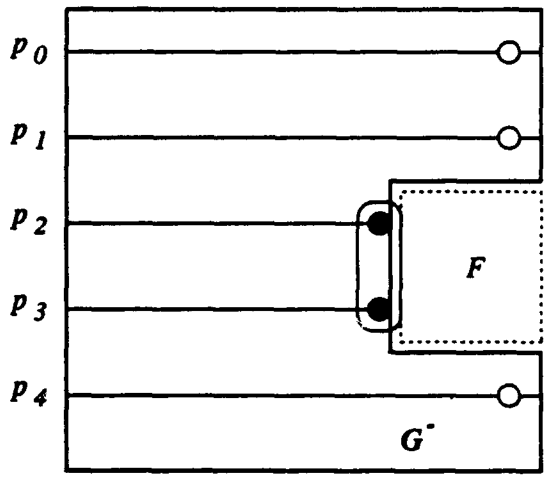

p_0
p_1
$c_{0,1}$
$c_{0,2}$
$c_{1,1}$
$c_{1,2}$


p_0
p_1
p_2
p_3
p_4
l_1
l_2
l_3
l_4
l_5
n_0
n_1
n_2
n_3
n_4
B
G
$\hat{G}$

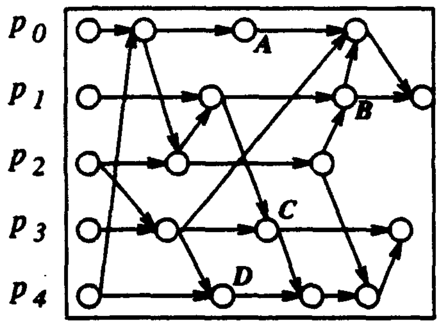

p_0
Current graph G
p_1
M_1
M_1'
L_2
M_2
p_2
p_3
p_4
Potential supergraph G'
p_0
Immediate supergraph $\hat{G} - B_1$
p_1
M_1
M_1'
L_2
B_2
p_2
p_3
p_4
B_1



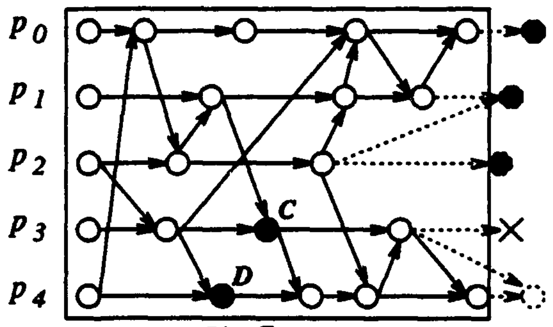
(a)



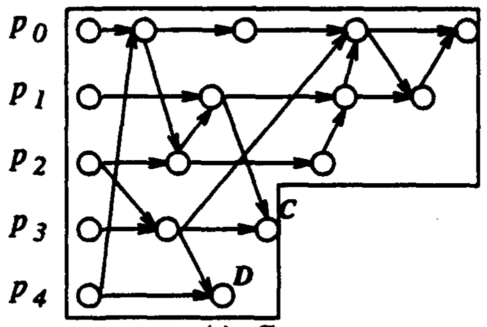
(b)



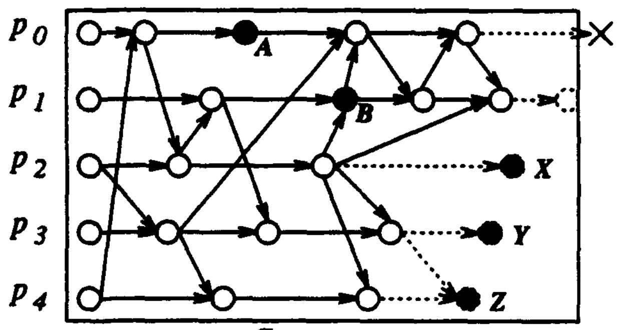
(a) G



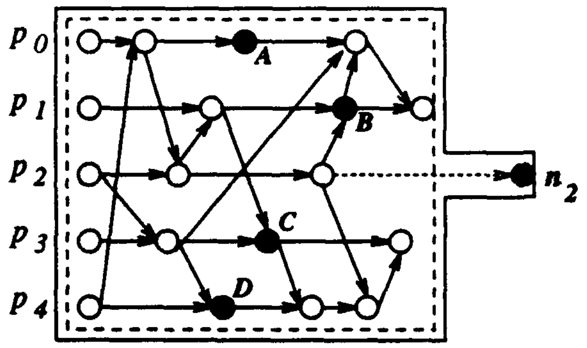
(b) G_b



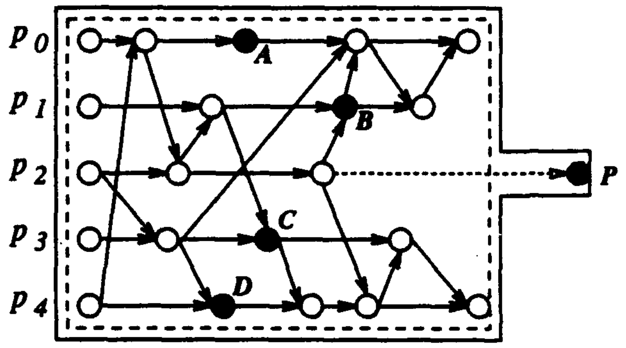
(c) G_c



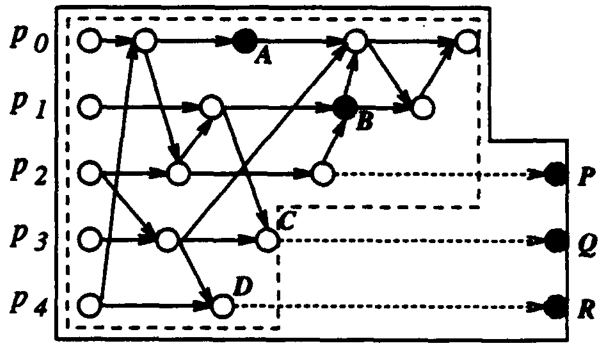
(d) G_d



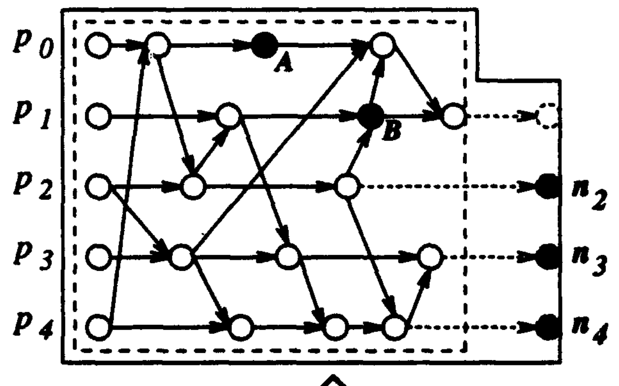
(e) G_e



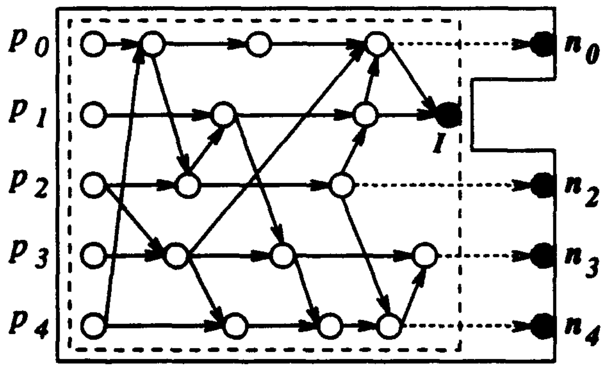
(f) G_f



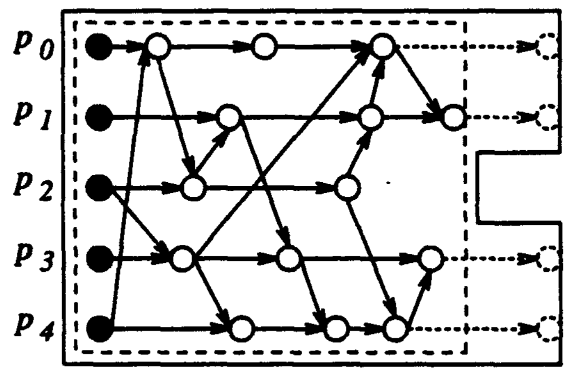
(g) G_g



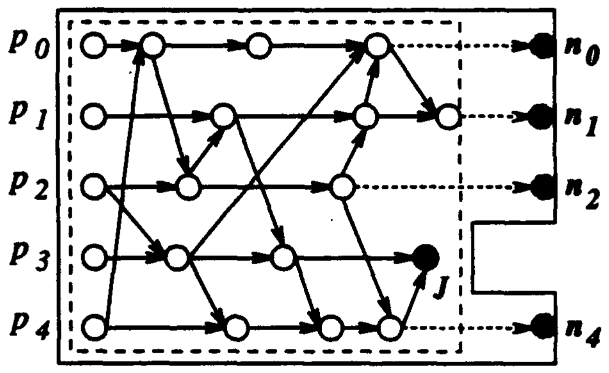
(a) $\hat{G} - n_0$



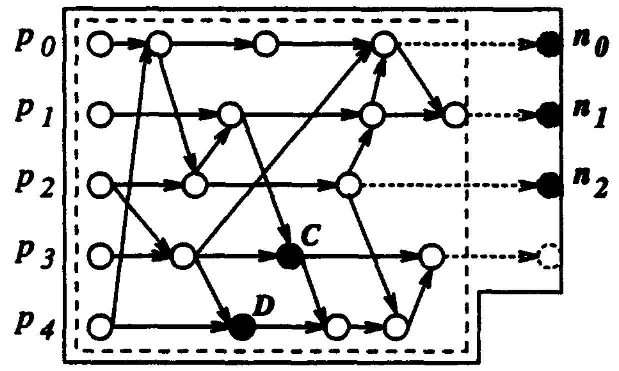
(b) $\hat{G} - n_1$



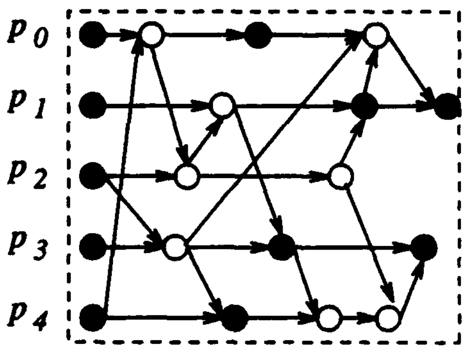
(c) $\hat{G} - n_2$



(d) $\hat{G} - n_3$

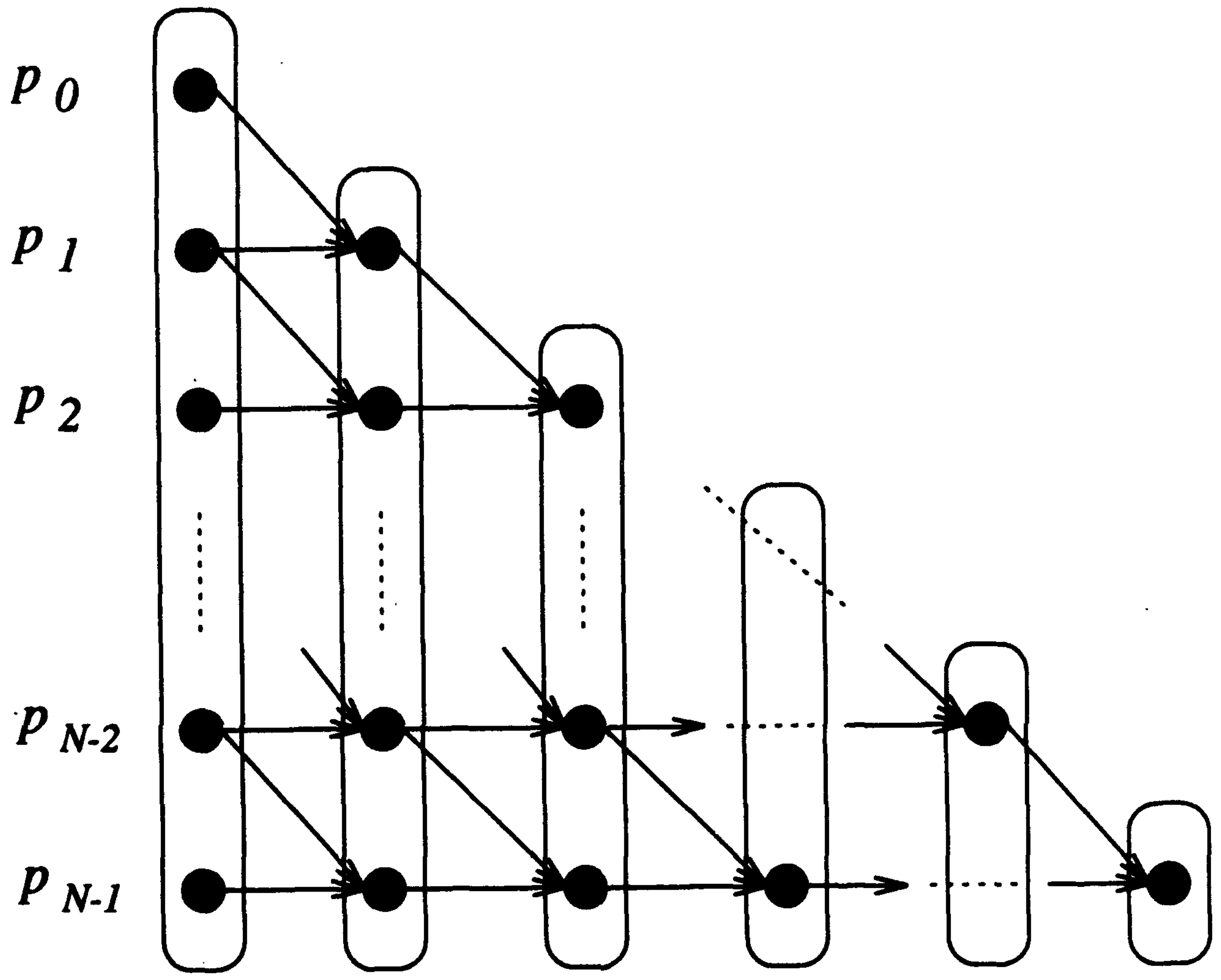


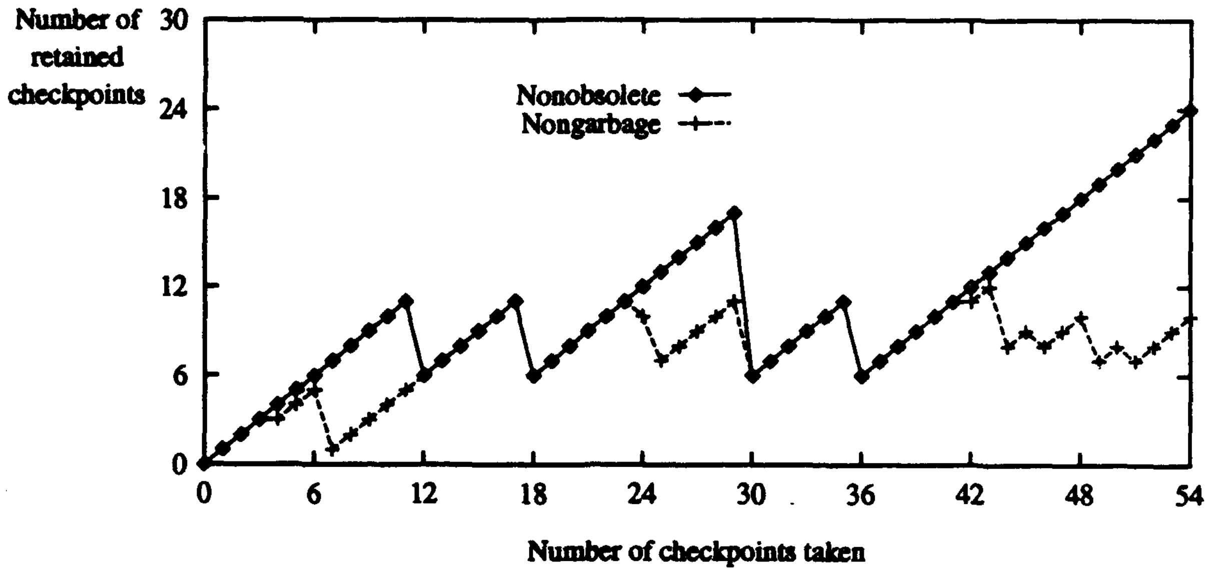
(e) $\hat{G} - n_4$



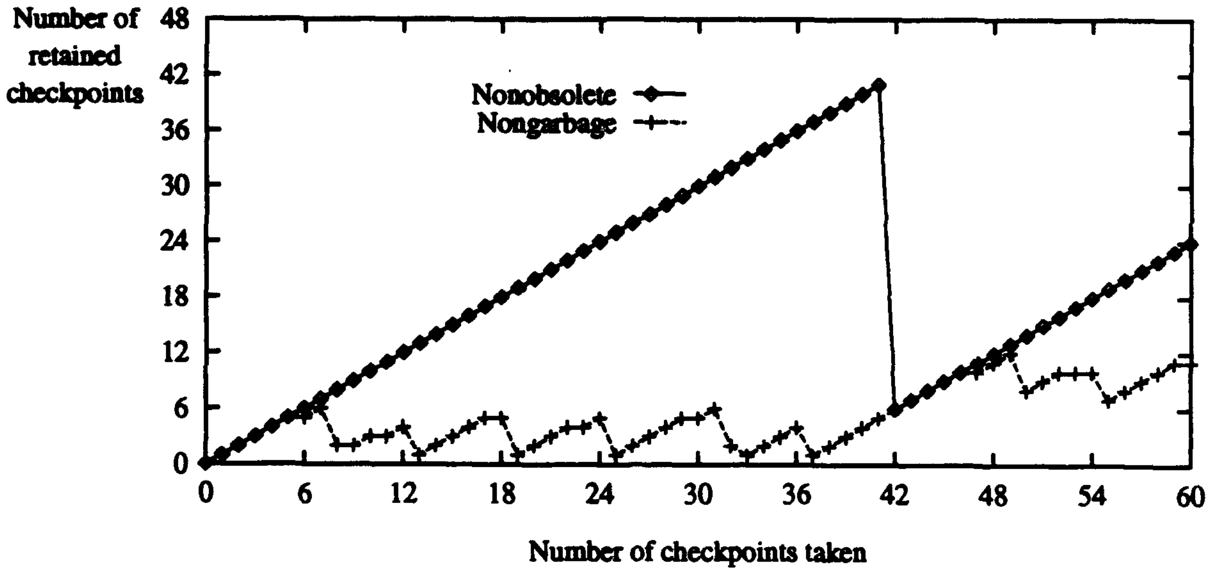
(f) G

```
/* $N_g(G)$ denotes the set of nongarbage checkpoints of $G$ */
/* $N$ is the number of processes */
/* $\hat{G}$ and $n_i$ are as defined in Fig. 5 */
for each $0 \leq i \leq N-1$ {
    apply the rollback propagation algorithm in Fig. 2 to the checkpoint graph $\hat{G} - n_i$
    to find the recovery line;
    all checkpoints in the recovery line except for the new-nodes are included in the set
    $N_g(G)$;
}
all of the checkpoints not in $N_g(G)$ can be garbage-collected.
```


p_0
p_1
p_2
p_{N-2}
p_{N-1}


Number of retained checkpoints
30
24
18
12
6
0
Nonobsolete
Nongarbage
0
6
12
18
24
30
36
42
48
54
Number of checkpoints taken

(c)


Number of retained checkpoints
48
42
36
30
24
18
12
6
0
Nonobsolete
Nongarbage
0
6
12
18
24
30
36
42
48
54
60
Number of checkpoints taken

(d)

Table 1. Execution and checkpoint parameters of the programs.

Benchmark programs	Cell Placement	Channel Router	Knight Tour	N-Queen
Number of processors	8	8	6	6
Machine	Intel iPSC/2 hypercube	Intel iPSC/2 hypercube	Encore Multimax	Encore Multimax
Execution time (sec)	322.7	469.3	273.2	1625.1
Checkpoint interval (sec)	35	40	30	150